

В этом задании вам предстоит решить задачу сегментации медицинских снимков. Часть кода с загрузкой данных написана за вас. Всю содержательную сторону вопроса вам нужно заполнить самостоятельно. Задание оценивается из 15 баллов.

Обратите внимание, что отчёт по заданию стоит целых 6 баллов. Он вынесен в отдельный пункт в конце тетради. Это сделано для того, чтобы тетрадь была оформлена как законченный документ о проведении экспериментов. Неотъемлемой составляющей отчёта является ответ на следующие вопросы:

- Что было сделано? Что получилось реализовать, что не получилось?
 - Какие результаты ожидалось получить?
 - Какие результаты были достигнуты?
 - Чем результаты различных подходов отличались друг от друга и от бейзлайна (если таковой присутствует)?
-

1. Для начала мы скачаем датасет: [ADDI project](#).
1. Разархивируем .rar файл.
2. Обратите внимание, что папка PH2 Dataset images должна лежать там же где и ipynb notebook.

Это фотографии двух типов **поражений кожи**: меланомы и родинки. В данном задании мы не будем заниматься их классификацией, а будем **сегментировать** их.

```
!wget -c https://www.dropbox.com/s/8lqrloi0mxj2acu/PH2Dataset.rar
--2022-06-23 08:14:29--
https://www.dropbox.com/s/8lqrloi0mxj2acu/PH2Dataset.rar
Resolving www.dropbox.com (www.dropbox.com)... 162.125.4.18,
2620:100:6019:18::a27d:412
Connecting to www.dropbox.com (www.dropbox.com)|162.125.4.18|:443...
connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/8lqrloi0mxj2acu/PH2Dataset.rar [following]
--2022-06-23 08:14:29--
https://www.dropbox.com/s/raw/8lqrloi0mxj2acu/PH2Dataset.rar
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location:
https://uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com/cd/0/
inline/
BnvdyvNuY9mKti02B06CfpB_U1MyfxE0ynbi743XG5znwo9pdAjIxYj_Smudgbuzan7Huu
FERQWbU02RUNGQ8jRoUzAFez_9iHi0hJ5WF_SHn-
FLPtgoX9LixhSpCThnb8PSeR8th0KU7BvUFdVm-goTSPL8pu5itkk-ohurV69ieA/file#
[following]
```

--2022-06-23 08:14:30--
https://uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com/cd/0/
inline/
BnvdyvNuY9mKti02B06CfpB_U1MyfxE0ynbi743XG5znwo9pdAjIxYj_Smudgbuzan7Huu
FErQWbU02RUNGQ8jRoUzAFez_9iHi0hJ5WF_SHn-
FLPtgoX9LixhSpCThnb8PSeR8th0KU7BvUFdVm-goTSPL8pu5itkk-ohurV69ieA/file
Resolving uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com
(uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com)...
162.125.6.15, 2620:100:6019:15::a27d:40f
Connecting to uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com
(uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com)|
162.125.6.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location:
/cd/0/inline2/BntBvXTkl3sWMX4JkYvst1d72MwjeQji1NIxy7i8iL_3TPgMIP_qN69F
I5BV8haJNBmAMDpBc0Kly-UEDeTWGeIf5Wvh8rvGaFGiPwmlK2M1sLXG6t5avhNpSKKv-
GfEt_BJNhfrvutMnLPZwhyk-
LK5X7DQw9PPSpLLAvpwkNXpmgShvvBSE8Qkq1GBtULIZ5b3TrGA07cl9bSV4bqUxVh0kX_
EKWuoa0NABLRQNRBEaQjtpv2QDC8nQ-
pFieDDfwx0SpZTCJUAfbESIDFSpGu4laEl4k8YXMwag77LKac85X2i1CvoBkNo3LwlvATI
4lm_n-P0SRU78gPAmcVit8NIZIou_4WzL-LMmWNsTq_uJG-
McRole0Eu_pQFvNGi_nTpkw77k1DQz3xsxWAWcAgTbw1Fy0c6X-IWE47CpngGmw/file
[following]
--2022-06-23 08:14:30--
https://uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com/cd/0/
inline2/
BntBvXTkl3sWMX4JkYvst1d72MwjeQji1NIxy7i8iL_3TPgMIP_qN69FI5BV8haJNBmAMD
pBc0Kly-UEDeTWGeIf5Wvh8rvGaFGiPwmlK2M1sLXG6t5avhNpSKKv-
GfEt_BJNhfrvutMnLPZwhyk-
LK5X7DQw9PPSpLLAvpwkNXpmgShvvBSE8Qkq1GBtULIZ5b3TrGA07cl9bSV4bqUxVh0kX_
EKWuoa0NABLRQNRBEaQjtpv2QDC8nQ-
pFieDDfwx0SpZTCJUAfbESIDFSpGu4laEl4k8YXMwag77LKac85X2i1CvoBkNo3LwlvATI
4lm_n-P0SRU78gPAmcVit8NIZIou_4WzL-LMmWNsTq_uJG-
McRole0Eu_pQFvNGi_nTpkw77k1DQz3xsxWAWcAgTbw1Fy0c6X-IWE47CpngGmw/file
Reusing existing connection to
uc8dfcec36b729618b95f1fccb3c.dl.dropboxusercontent.com:443.
HTTP request sent, awaiting response... 200 OK
Length: 161895206 (154M) [application/rar]
Saving to: 'PH2Dataset.rar'

PH2Dataset.rar 100%[=====>] 154.39M 101MB/s in
1.5s

2022-06-23 08:14:32 (101 MB/s) - 'PH2Dataset.rar' saved
[161895206/161895206]

!ls

PH2Dataset.rar sample_data

```
get_ipython().system_raw("unrar x PH2Dataset.rar")
```

Структура датасета у нас следующая:

```
IMD_002/  
  IMD002_Dermoscopic_Image/  
    IMD002.bmp  
  IMD002_lesion/  
    IMD002_lesion.bmp  
  IMD002_roi/  
  ...  
IMD_003/  
  ...  
  ...
```

Здесь `X.bmp` — изображение, которое нужно сегментировать, `X_lesion.bmp` — результат сегментации.

Для загрузки датасета можно использовать `skimage.io.imread()`

```
images = []  
lesions = []  
from skimage.io import imread  
import os  
from tqdm import tqdm  
root = 'PH2Dataset'  
  
for root, dirs, files in os.walk(os.path.join(root, 'PH2 Dataset  
images')):  
    if root.endswith('_Dermoscopic_Image'):  
        images.append(imread(os.path.join(root, files[0])))  
    if root.endswith('_lesion'):  
        lesions.append(imread(os.path.join(root, files[0])))
```

Изображения имеют разные размеры. Давайте изменим их размер на 256×256 пикселей. Для изменения размера изображений можно использовать `skimage.transform.resize()`. Эта функция также автоматически нормализует изображения в диапазоне $[0, 1]$.

```
from skimage.transform import resize  
size = (256, 256)  
X = [resize(x, size, mode='constant', anti_aliasing=True,) for x in  
images]  
Y = [resize(y, size, mode='constant', anti_aliasing=False) > 0.5 for y  
in lesions]  
  
import numpy as np  
X = np.array(X, np.float32)  
Y = np.array(Y, np.float32)  
print(f'Loaded {len(X)} images')
```



```

data_val = DataLoader(list(zip(np.rollaxis(X[val], 3, 1), Y[val,
np.newaxis])),
                        batch_size=batch_size, shuffle=True)
data_ts = DataLoader(list(zip(np.rollaxis(X[ts], 3, 1), Y[ts,
np.newaxis])),
                        batch_size=batch_size, shuffle=True)

import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

cuda

```

Реализация различных архитектур:

Ваше задание будет состоять в том, чтобы написать несколько нейросетевых архитектур для решения задачи семантической сегментации. Сравнить их по качеству на тесте и испробовать различные лосс функции для них.

SegNet [2 балла]

- Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). [SegNet: A deep convolutional encoder-decoder architecture for image segmentation](#)

Внимательно посмотрите из чего состоит модель и для чего выбраны те или иные блоки.

```

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models
import torch.optim as optim
from time import time

from matplotlib import rcParams
rcParams['figure.figsize'] = (15,4)
import matplotlib
font = {'family' : 'monospace',
        'weight' : 'bold',
        'size'   : 10}

matplotlib.rc('font', **font)

```

```

from torch.nn.modules import activation
class CustomConvLayer(nn.Module):
    def __init__(self, in_channels: int, out_channels: int,
                  kernel_size: int, stride: int=1, padding: int=1,
inplace: bool=False):
        super(CustomConvLayer, self).__init__()
        self.layer = nn.Sequential(
            nn.Conv2d(in_channels, out_channels,
kernel_size=kernel_size, stride=stride, padding=padding),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=inplace)
        )

    def forward(self, input: torch.Tensor):
        output = self.layer(input)
        return output

class Custom2LayersCNN(nn.Module):
    def __init__(self, in_channels: int, out_channels: int,
                  kernel_size: int, stride: int=1, padding: int=1,
inplace: bool=False):
        super(Custom2LayersCNN, self).__init__()
        self.layer = nn.Sequential(
            CustomConvLayer(in_channels=in_channels,
out_channels=out_channels,
kernel_size=kernel_size, stride=stride,
padding=padding,
inplace=inplace),
            CustomConvLayer(in_channels=out_channels,
out_channels=out_channels,
kernel_size=kernel_size, stride=stride,
padding=padding,
inplace=inplace)
        )

    def forward(self, input: torch.Tensor):
        output = self.layer(input)
        return output

class Custom3LayersCNN(nn.Module):
    def __init__(self, in_channels: int, out_channels: int,
                  kernel_size: int, stride: int=1, padding: int=1,
inplace: bool=False):
        super(Custom3LayersCNN, self).__init__()
        self.layer = nn.Sequential(
            CustomConvLayer(in_channels=in_channels,
out_channels=out_channels,
kernel_size=kernel_size, stride=stride,
padding=padding,

```

```

        inplace=inplace),
        CustomConvLayer(in_channels=out_channels,
out_channels=out_channels,
        kernel_size=kernel_size, stride=stride,
padding=padding,
        inplace=inplace),
        CustomConvLayer(in_channels=out_channels,
out_channels=out_channels,
        kernel_size=kernel_size, stride=stride,
padding=padding,
        inplace=inplace)
    )

```

```

def forward(self, input: torch.Tensor):
    output = self.layer(input)
    return output

```

```

class SegNet(nn.Module):
    def __init__(self):
        super().__init__()
        # encoder (downsampling)
        self.enc_conv0 = Custom2LayersCNN(in_channels=3,
out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool0 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 256 -> 128
        self.enc_conv1 = Custom2LayersCNN(in_channels=32,
out_channels=64, kernel_size=3, stride=1, padding=1)
        self.pool1 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 128 -> 64
        self.enc_conv2 = Custom3LayersCNN(in_channels=64,
out_channels=128, kernel_size=3, stride=1, padding=1)
        self.pool2 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 64 -> 32
        self.enc_conv3 = Custom3LayersCNN(in_channels=128,
out_channels=256, kernel_size=3, stride=1, padding=1)
        self.pool3 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 32 -> 16
        # bottleneck
        self.bottleneck_conv = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3,
stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3,
stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU()
        )
        # decoder (upsampling)
        self.upsample0 = torch.nn.MaxUnpool2d(2, 2) # 16 -> 32

```

```

        self.dec_conv0 = Custom3LayersCNN(in_channels=256,
out_channels=128, kernel_size=3, stride=1, padding=1)
        self.upsample1 = torch.nn.MaxUnpool2d(2, 2) # 32 -> 64
        self.dec_conv1 = Custom3LayersCNN(in_channels=128,
out_channels=64, kernel_size=3, stride=1, padding=1)
        self.upsample2 = torch.nn.MaxUnpool2d(2, 2) # 64 -> 128
        self.dec_conv2 = Custom2LayersCNN(in_channels=64,
out_channels=32, kernel_size=3, stride=1, padding=1)
        self.upsample3 = torch.nn.MaxUnpool2d(2, 2) # 128 -> 256
        self.dec_conv3 = Custom2LayersCNN(in_channels=32,
out_channels=1, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        # encoder
        e0, indices0 = self.pool0(self.enc_conv0(x))
        e1, indices1 = self.pool1(self.enc_conv1(e0))
        e2, indices2 = self.pool2(self.enc_conv2(e1))
        e3, indices3 = self.pool3(self.enc_conv3(e2))
        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        d0 = self.dec_conv0(self.upsample0(b, indices3))
        d1 = self.dec_conv1(self.upsample1(d0, indices2))
        d2 = self.dec_conv2(self.upsample2(d1, indices1))
        d3 = self.dec_conv3(self.upsample3(d2, indices0)) # no
activation
        return d3

```

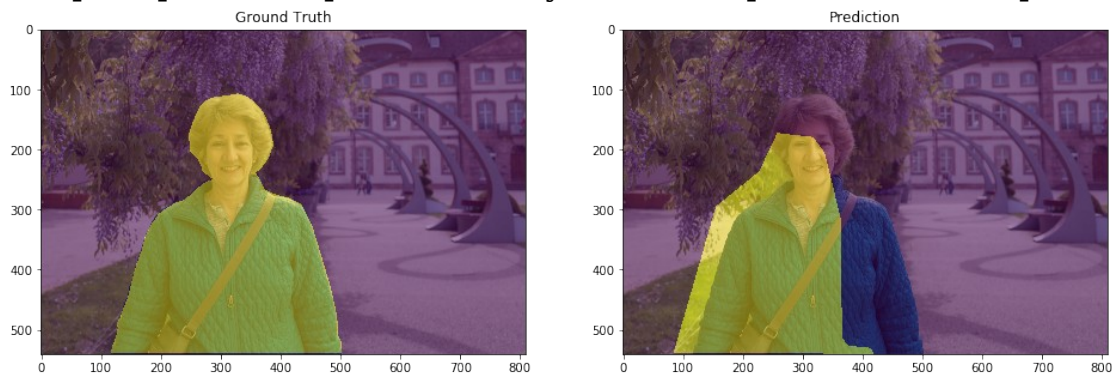
Метрика

В данном разделе предлагается использовать следующую метрику для оценки качества:

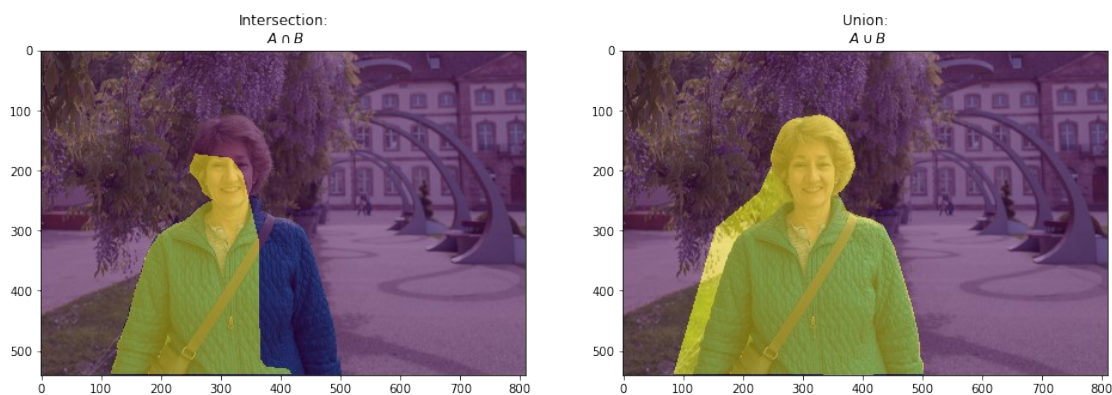
$$IoU = \frac{\text{target} \cap \text{prediction}}{\text{target} \cup \text{prediction}}$$

Пересечение ($A \cap B$) состоит из пикселей, найденных как в маске предсказания, так и в основной маске истины, тогда как объединение ($A \cup B$) просто состоит из всех пикселей, найденных либо в маске предсказания, либо в целевой маске.

Для примера посмотрим на истину (слева) и предсказание (справа):



Тогда пересечение и объединение будет выглядеть так:



```
def iou_pytorch(outputs: torch.Tensor, labels: torch.Tensor):
    # You can comment out this line if you are passing tensors of
    # equal shape
    # But if you are passing output from UNet or something it will
    # most probably
    # be with the BATCH x 1 x H x W shape
    outputs = outputs.squeeze(1).byte() # BATCH x 1 x H x W => BATCH
    x H x W
    labels = labels.squeeze(1).byte()
    SMOOTH = 1e-8
    intersection = (outputs & labels).float().sum((1, 2)) # Will be
    zero if Truth=0 or Prediction=0
    union = (outputs | labels).float().sum((1, 2)) # Will be
    zzero if both are 0

    iou = (intersection + SMOOTH) / (union + SMOOTH) # We smooth our
    devision to avoid 0/0

    thresholded = torch.clamp(20 * (iou - 0.5), 0, 10).ceil() / 10 #
    This is equal to comparing with thresholds

    return thresholded #
```

Функция потерь [1 балл]

Не менее важным, чем построение архитектуры, является определение **оптимизатора и функции потерь**.

Функция потерь - это то, что мы пытаемся минимизировать. Многие из них могут быть использованы для задачи бинарной семантической сегментации.

Популярным методом для бинарной сегментации является *бинарная кросс-энтропия*, которая задается следующим образом:

$$L_{BCE}(y, \hat{y}) = - \sum_i \left[y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log (1 - \sigma(\hat{y}_i)) \right].$$

где y это таргет желаемого результата и \hat{y} является выходом модели. σ - это *логистическая функция*, который преобразует действительное число R в вероятность $[0, 1]$.

Однако эта потеря страдает от проблем численной нестабильности. Самое главное, что $\lim_{x \rightarrow 0} \log(x) = \infty$ приводит к неустойчивости в процессе оптимизации. Рекомендуется посмотреть следующее *упрощение*. Эта функция эквивалентна первой и не так подвержена численной неустойчивости:

$$L_{BCE} = \hat{y} - y\hat{y} + \log(1 + \exp(-\hat{y})).$$

```
def bce_loss(y_real: torch.Tensor, y_pred: torch.Tensor):  
    return torch.mean(y_pred - y_real*y_pred + torch.log(1+torch.exp(-  
y_pred)))
```

Тренировка [1 балл]

Мы определим цикл обучения в функции, чтобы мы могли повторно использовать его.

```
from collections.abc import Iterable, Callable  
  
def score_model(model, metric, data):  
    model.eval() # testing mode  
    scores = 0  
    for X_batch, Y_label in data:  
        Y_pred = torch.sigmoid(model(X_batch.to(device))) > 0.5  
        scores += metric(Y_pred, Y_label.to(device)).mean().item()  
  
    return scores/len(data)  
  
def train(model: nn.Module, opt: torch.optim, loss_fn, epochs: int,  
          data_tr: Iterable, data_val: Iterable, scheduler:  
torch.optim.lr_scheduler=None,  
          metric=iou_pytorch, score_model: Callable=score_model):
```

```

X_val, Y_val = next(iter(data_val))
history = []
for epoch in range(epochs):
    with tqdm(data_tr, unit="batch") as tepoch:
        tepoch.set_description(f"Epoch {epoch + 1}")
        tic = time()
        # print('* Epoch %d/%d' % (epoch+1, epochs))
        log_template = "\nEpoch {ep:03d} train_loss: {t_loss:0.4f}
train_iou: {t_iou:0.4f} val_loss: {v_loss:0.4f} val_iou: {v_iou:0.4f}"
        avg_iou_train = 0
        avg_loss_train = 0
        avg_loss_val = 0
        model.train() # train mode
        for X_batch, Y_batch in tepoch:
            X_batch = X_batch.to(device)
            Y_batch = Y_batch.to(device)

            opt.zero_grad()
            # set parameter gradients to zero
            # forward
            Y_pred = model(X_batch)
            iou = metric(torch.sigmoid(Y_pred) > 0.5,
Y_batch).mean().item()
            loss = loss_fn(Y_batch, Y_pred) # forward-pass
            loss.backward() # backward-pass
            opt.step() # update weights
            if scheduler:
                scheduler.step()
            # calculate loss to show the user
            avg_loss_train += loss / len(data_tr)
            avg_iou_train += iou / len(data_tr)
        toc = time()
        # print('loss: %f' % avg_loss)
        # show intermediate results
        model.eval() # testing mode
        Y_hat = model(X_val.to(device)).detach().cpu().numpy()
        iou_val = score_model(model, metric, data_val)
        # validation loss
        for X_batch, Y_batch in data_val:
            X_batch = X_batch.to(device)
            Y_batch = Y_batch.to(device)
            with torch.set_grad_enabled(False):
                Y_pred = model(X_batch)
                avg_loss_val += loss_fn(Y_batch, Y_pred) /
len(data_val)
            history.append((avg_loss_train.item(), avg_iou_train,
avg_loss_val.item(), iou_val))
        # Visualize tools
        clear_output(wait=True)
        for k in range(6):

```

```

        plt.subplot(2, 6, k+1)
        plt.imshow(np.rollaxis(X_val[k].numpy(), 0, 3),
cmap='gray')
        plt.title('Real')
        plt.axis('off')

        plt.subplot(2, 6, k+7)
        plt.imshow(Y_hat[k, 0], cmap='gray')
        plt.title('Output')
        plt.axis('off')
    plt.suptitle('%d / %d - loss: %f' % (epoch+1, epochs,
avg_loss_train))
    plt.show()
    tepoch.update(1)
    tqdm.write(log_template.format(ep=epoch+1,
t_loss=avg_loss_train,
                                t_iou=avg_iou_train,
v_loss=avg_loss_val, v_iou=iou_val))
    torch.cuda.empty_cache()
    return history

```

Инференс [1 балл]

После обучения модели эту функцию можно использовать для прогнозирования сегментации на новых данных:

```

def predict(model, data):
    model.eval() # testing mode
    Y_pred = [model(X_batch.to(device)).detach().cpu().numpy() for
X_batch, _ in data]
    return np.array(Y_pred)

def score_model(model, metric, data):
    model.eval() # testing mode
    scores = 0
    for X_batch, Y_label in data:
        Y_pred = torch.sigmoid(model(X_batch.to(device))) > 0.5
        scores += metric(Y_pred, Y_label.to(device)).mean().item()

    return scores/len(data)

```

Основной момент: обучение

Обучите вашу модель. Обратите внимание, что обучать необходимо до сходимости. Если указанного количества эпох (20) не хватило, попробуйте изменять количество эпох до сходимости алгоритма. Сходимость определяйте по изменению функции потерь на валидационной выборке. С параметрами оптимизатора можно спокойно играть, пока вы не найдете лучший вариант для себя.

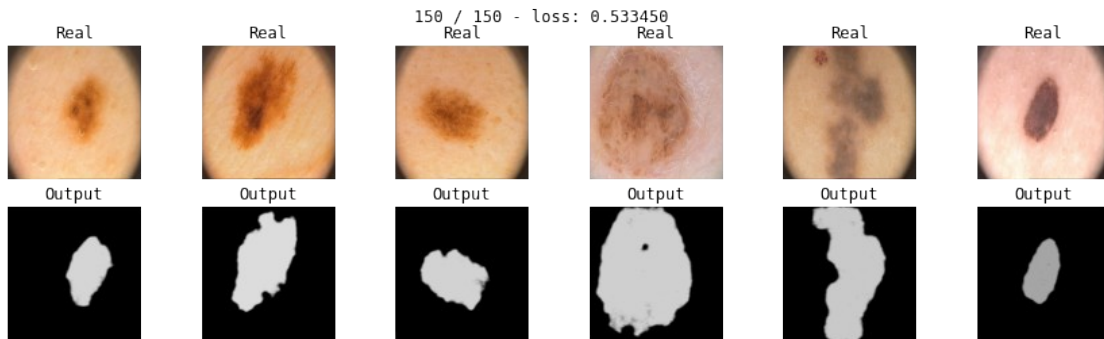
```

model = SegNet().to(device)

histories = {}
scores = {}

max_epochs = 150
opt = optim.AdamW(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
histories[bce_loss.__name__] = train(model, opt, bce_loss, max_epochs,
data_tr, data_val)

```



```

Epoch 150 train_loss: 0.5334 train_iou: 0.9250 val_loss: 0.5550
val_iou: 0.7360

```

```

scores[bce_loss.__name__] = score_model(model, iou_pytorch, data_val)

def loss_experiment(loss_fn: Callable, model: nn.Module, data_tr:
Iterable, data_val: Iterable, opt=None,
                    scheduler=None, lr: float=1e-4, step_size: int=10,
                    gamma: float=0.5, epochs: int=150) -> dict:
    assert loss_fn.__name__, 'Check loss function'
    global histories, scores
    opt = opt if opt else optim.AdamW(model.parameters(), lr=lr)
    scheduler = scheduler if scheduler else
optim.lr_scheduler.StepLR(opt, step_size=step_size, gamma=gamma)
    histories[loss_fn.__name__] = train(model, opt, loss_fn,
max_epochs, data_tr, data_val)
    scores[loss_fn.__name__] = score_model(model, iou_pytorch,
data_val)
    return histories

def plot_histories(histories: list):
    plt.style.use('seaborn-dark-palette')
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20,6))
    for loss_fn_name, history in histories.items():
        loss, acc, val_loss, val_acc = zip(*history)
        ax[0].plot(loss, label=f'train_{loss_fn_name}')
        ax[0].plot(val_loss, label=f'val_{loss_fn_name}')

```

```

ax[1].plot(acc, label=f'train_{loss_fn_name}_iou')
ax[1].plot(val_acc, label=f'val_{loss_fn_name}_iou')
ax[0].set_title('Loss')
ax[1].set_title('Intersection over Union (IoU)')

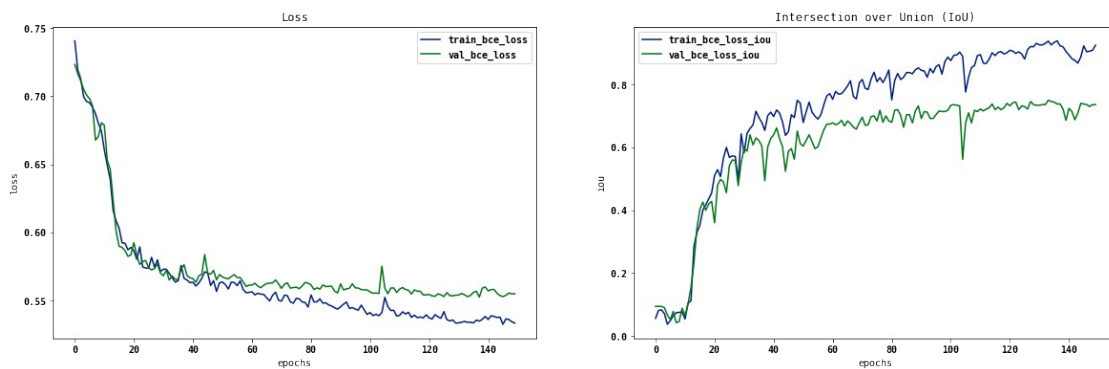
ax[0].legend(loc='best')
ax[1].legend(loc='best')

ax[0].set_xlabel("epochs")
ax[0].set_ylabel("loss")

ax[1].set_xlabel("epochs")
ax[1].set_ylabel("iou")
plt.show()

plot_histories(histories)

```



Ответьте себе на вопрос: не переобучается ли моя модель?

Дополнительные функции потерь [2 балла]

В данном разделе вам потребуется имплементировать две функции потерь: DICE и Focal loss. Если у вас что-то не учится, велика вероятность, что вы ошиблись или учите слишком мало эпох, прежде чем бить тревогу попробуйте перебрать различные варианты и убедитесь, что во всех других сетапах сеть достигает желанного результата. СПОЙЛЕР: учиться она будет при всех лоссах, предложенных в этом задании.

1. Dice coefficient: Учитывая две маски X и Y , общая метрика для измерения расстояния между этими двумя масками задается следующим образом:

$$D(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

Эта функция не является дифференцируемой, но это необходимое свойство для градиентного спуска. В данном случае мы можем приблизить его с помощью:

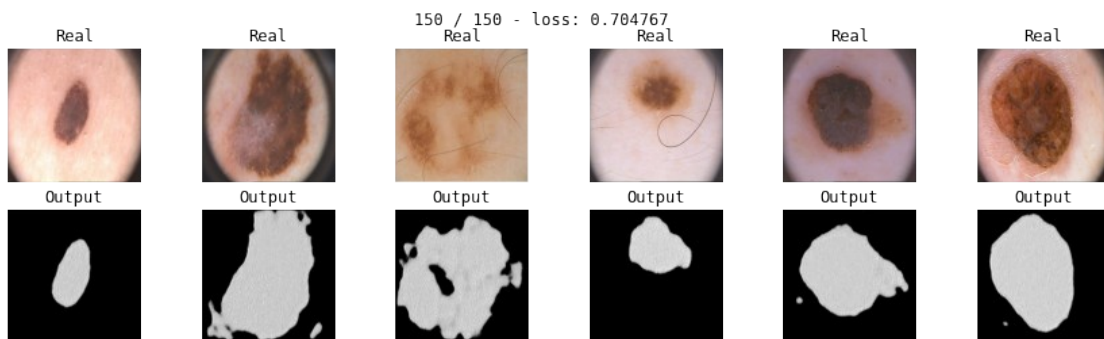
$$L_D(X, Y) = 1 - \frac{1}{256 \times 256} \times \frac{\sum_i 2 X_i Y_i}{\sum_i X_i + Y_i}.$$

Не забудьте подумать о численной неустойчивости, возникающей в математической формуле.

```
def dice_loss(y_real: torch.Tensor, y_pred: torch.Tensor, eps:
float=1e-8) -> float:
    y_pred = torch.sigmoid(y_pred)
    res = 1 - torch.mean(2 * (y_pred * y_real + eps) / ((y_pred +
y_real + eps)))
    return res
```

Проводим тестирование:

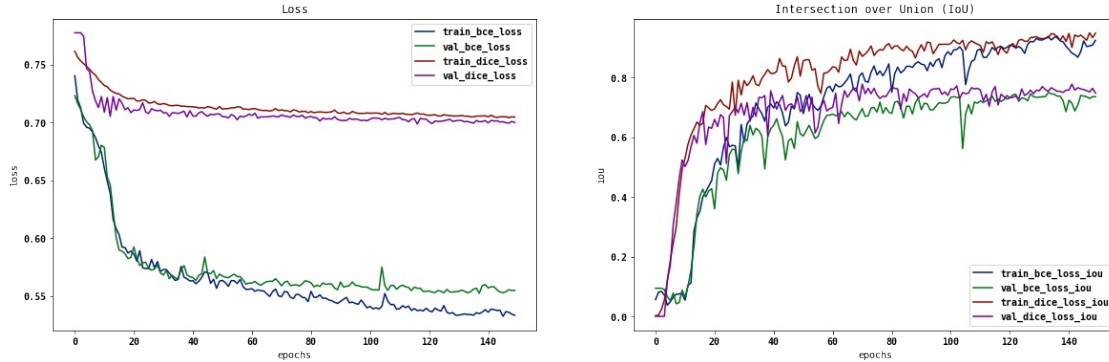
```
model_dice = SegNet().to(device)
max_epochs = 150
opt = optim.AdamW(model_dice.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
histories = loss_experiment(loss_fn=dice_loss, model=model_dice,
data_tr=data_tr, data_val=data_val, opt=opt,
scheduler=scheduler)
```



```
Epoch 150 train_loss: 0.7048 train_iou: 0.9490 val_loss: 0.7001
val_iou: 0.7480
```

```
scores[dice_loss.__name__] = score_model(model_dice, iou_pytorch,
data_val)
```

```
plot_histories(histories)
```

2. Focal loss:

Окей, мы уже с вами умеем делать BCE loss:

$$L_{BCE}(y, \hat{y}) = - \sum_i \left[y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log (1 - \sigma(\hat{y}_i)) \right].$$

Проблема с этой потерей заключается в том, что она имеет тенденцию приносить пользу классу **большинства** (фоновому) по отношению к классу **меньшинства** (переднему). Поэтому обычно применяются весовые коэффициенты к каждому классу:

$$L_{wBCE}(y, \hat{y}) = - \sum_i \alpha_i \left[y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log (1 - \sigma(\hat{y}_i)) \right].$$

Традиционно вес α_i определяется как обратная частота класса этого пикселя i , так что наблюдения миноритарного класса весят больше по отношению к классу большинства.

Еще одним недавним дополнением является взвешенный пиксельный вариант, которая взвешивает каждый пиксель по степени уверенности, которую мы имеем в предсказании этого пикселя.

$$L_{focal}(y, \hat{y}) = - \sum_i \left[(1 - \sigma(\hat{y}_i))^\gamma y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log (1 - \sigma(\hat{y}_i)) \right].$$

Зафиксируем значение $\gamma=2$.

```
def focal_loss(y_real:torch.Tensor, y_pred:torch.Tensor, eps:
float=1e-7, gamma: float=2) -> float:
    y_pred = torch.clamp(torch.sigmoid(y_pred), min=eps, max=1-eps) #
    hint: torch.clamp
    loss = -torch.mean((1 - y_pred)**gamma * y_real * torch.log(y_pred
+ eps) + (1 - y_real) * torch.log(1 - y_pred + eps))
    return loss

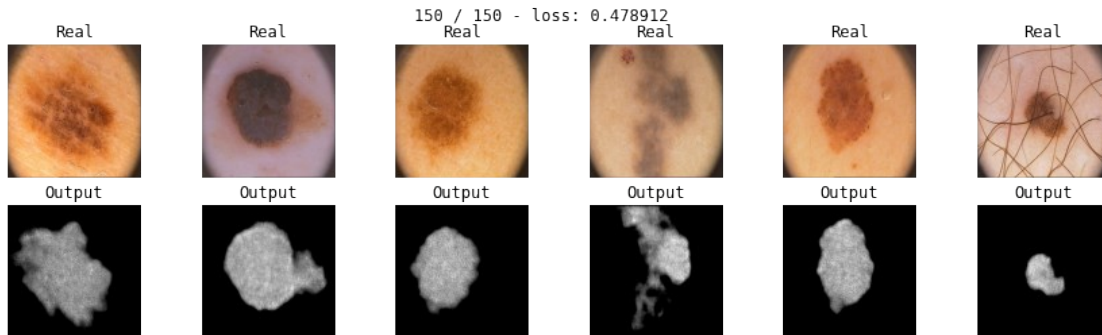
model_focal = SegNet().to(device)
max_epochs = 150
```



```

opt = optim.AdamW(model_focal.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
histories = loss_experiment(loss_fn=focal_loss, model=model_focal,
data_tr=data_tr, data_val=data_val, opt=opt,
scheduler=scheduler)

```



Epoch 150 train_loss: 0.4789 train_iou: 0.8730 val_loss: 0.4822
val_iou: 0.7060

```

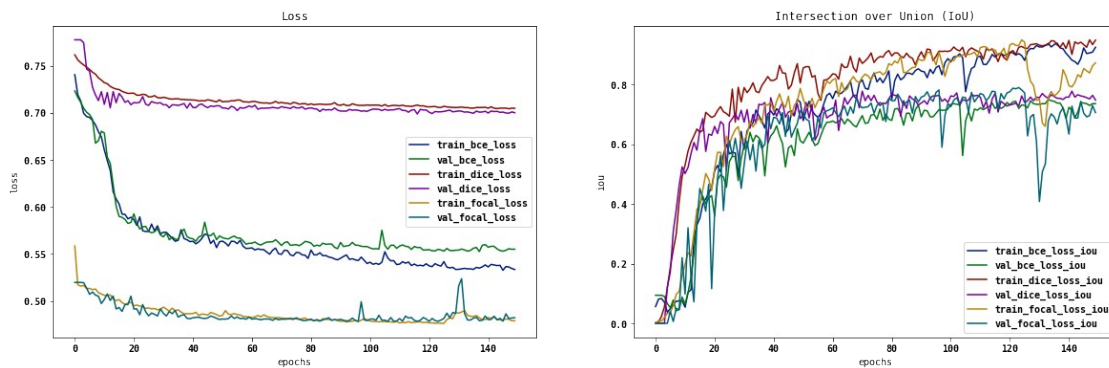
scores[focal_loss.__name__] = score_model(model_focal, iou_pytorch,
data_val)

```

```

plot_histories(histories)

```



[BONUS] Мир сегментационных лоссов [5 баллов]

В данном блоке предлагаем вам написать одну функцию потерь самостоятельно. Для этого необходимо прочитать статью и имплементировать ее. Кроме того провести численное сравнение с предыдущими функциями. Какие варианты?

1) Можно учесть Total Variation 2) Lova 3) BCE но с Soft Targets (что-то типа label-smoothing для многослассовой классификации) 4) Любой другой

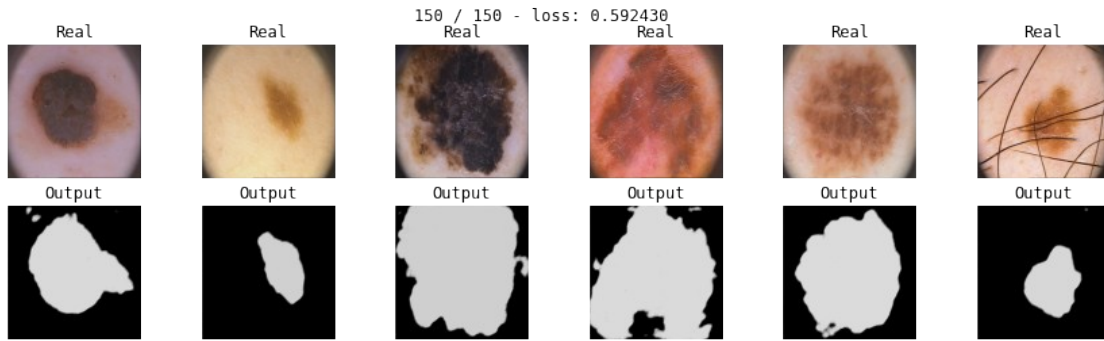
- [Physiological Inspired Deep Neural Networks for Emotion Recognition". IEEE Access, 6, 53930-53943.](#)
- [Boundary loss for highly unbalanced segmentation](#)
- [Tversky loss function for image segmentation using 3D fully convolutional deep networks](#)
- [Correlation Maximized Structural Similarity Loss for Semantic Segmentation](#)
- [Topology-Preserving Deep Image Segmentation](#)

Так как Тверский лосс очень похож на данные выше, то за него будет проставлено только 3 балла (при условии, если в модели нет ошибок при обучении). Постарайтесь сделать что-то интереснее.

```
def tversky_loss(y_real, y_pred, alpha=0.2, beta=0.8, eps = 1e-8):
    y_pred = torch.sigmoid(y_pred)
    res = 1 - (torch.sum(y_pred * y_real + eps)) / (eps +
    torch.sum(y_pred * y_real) +
    alpha * torch.sum((1 -
    y_real) * y_pred) +
    beta * torch.sum(y_real *
    (1 - y_pred)))
    return res

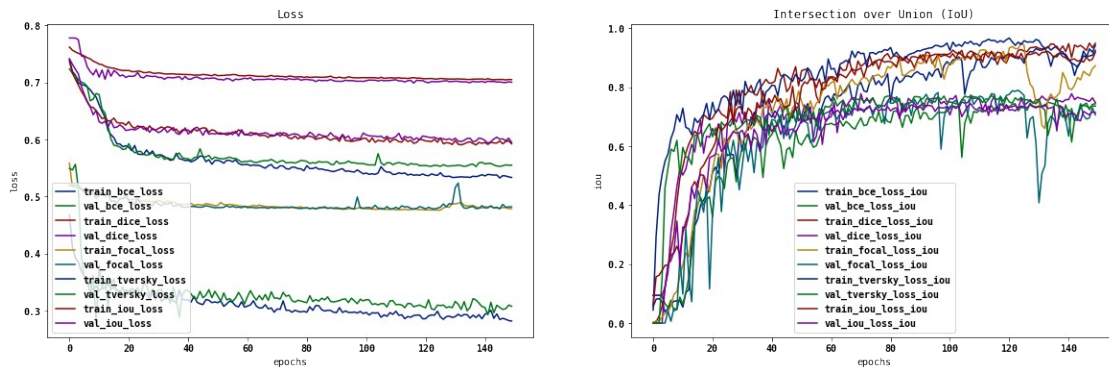
def iou_loss(y_real, y_pred, eps = 1e-8):
    y_pred = torch.clamp(torch.sigmoid(y_pred), min=eps, max=1-eps)
    res = 1 - torch.sum(y_pred * y_real) / (eps + torch.sum(y_pred) +
    torch.sum(y_real) - torch.sum(y_pred * y_real))
    return res

losses = [tversky_loss, iou_loss]
for loss in losses:
    model_custom = SegNet().to(device)
    max_epochs = 150
    opt = optim.AdamW(model_custom.parameters(), lr=1e-4)
    scheduler = optim.lr_scheduler.StepLR(opt, step_size=10,
    gamma=0.5)
    loss_experiment(loss_fn=loss, model=model_custom, data_tr=data_tr,
    data_val=data_val, opt=opt,
    scheduler=scheduler)
    print(f"Model with {loss.__name__} :", score_model(model_custom,
    iou_pytorch, data_val))
```



Epoch 150 train_loss: 0.5924 train_iou: 0.9190 val_loss: 0.5946
 val_iou: 0.7140
 Model with iou_loss : 0.7139999866485596

plot_histories(histories)



scores

```
{'bce_loss': 0.7360000252723694,
 'dice_loss': 0.7480000138282776,
 'focal_loss': 0.706000030040741,
 'iou_loss': 0.7139999985694885,
 'tversky_loss': 0.7440000176429749}
```

```
segnet_info = {
    'name': "segnet",
    'history': histories,
    'scores': scores
}
```

```
import json
with open("segnet.json", "w") as outfile:
    json.dump(segnet_info, outfile)
```

U-Net [2 балла]

U-Net — это архитектура нейронной сети, которая получает изображение и выводит его. Первоначально он был задуман для семантической сегментации (как мы ее будем использовать), но он настолько успешен, что с тех пор используется в других контекстах. Получая на вход медицинское изображение, он выведет изображение в оттенках серого, где интенсивность каждого пикселя зависит от вероятности того, что этот пиксель принадлежит интересующей нас области.

У нас в архитектуре все так же существует энкодер и декодер, как в **SegNet**, но отличительной особенностью данной модели являются *skip-connections*, соединяющие части декодера и энкодера. То есть для того чтобы передать на вход декодера тензор, мы конкатенируем симметричный выход с энкодера и выход предыдущего слоя декодера.

- Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

```
class UNet(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder (downsampling)
        # Each enc_conv/dec_conv block should look like this:
        # nn.Sequential(
        #     nn.Conv2d(...),
        #     ... (2 or 3 conv layers with relu and batchnorm),
        # )
        self.enc_conv0 = Custom2LayersCNN(in_channels=3,
out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool0 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 256 -> 128
        self.enc_conv1 = Custom2LayersCNN(in_channels=32,
out_channels=64, kernel_size=3, stride=1, padding=1)
        self.pool1 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 128 -> 64
        self.enc_conv2 = Custom3LayersCNN(in_channels=64,
out_channels=128, kernel_size=3, stride=1, padding=1)
        self.pool2 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 64 -> 32
        self.enc_conv3 = Custom3LayersCNN(in_channels=128,
out_channels=256, kernel_size=3, stride=1, padding=1)
        self.pool3 = torch.nn.MaxPool2d(2, 2, ceil_mode=True,
return_indices=True) # 32 -> 16

        # bottleneck
```

```

        self.bottleneck_conv = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3,
stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3,
stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU()
        )

        # decoder (upsampling)
        self.upsample0 = torch.nn.MaxUnpool2d(2, 2) # 16 -> 32
        self.dec_conv0 = Custom3LayersCNN(in_channels=256*2,
out_channels=128, kernel_size=3, stride=1, padding=1)
        self.upsample1 = torch.nn.MaxUnpool2d(2, 2) # 32 -> 64
        self.dec_conv1 = Custom3LayersCNN(in_channels=128*2,
out_channels=64, kernel_size=3, stride=1, padding=1)
        self.upsample2 = torch.nn.MaxUnpool2d(2, 2) # 64 -> 128
        self.dec_conv2 = Custom3LayersCNN(in_channels=64*2,
out_channels=32, kernel_size=3, stride=1, padding=1)
        self.upsample3 = torch.nn.MaxUnpool2d(2, 2) # 128 -> 256
        self.dec_conv3 = Custom3LayersCNN(in_channels=32*2,
out_channels=1, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        # encoder
        e0, indices0 = self.pool0(self.enc_conv0(x))
        e1, indices1 = self.pool1(self.enc_conv1(e0))
        e2, indices2 = self.pool2(self.enc_conv2(e1))
        e3, indices3 = self.pool3(self.enc_conv3(e2))
        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        d0 = self.dec_conv0(self.upsample0(torch.cat([e3, b], dim=1),
torch.cat([indices3, indices3], dim=1)))
        d1 = self.dec_conv1(self.upsample1(torch.cat([e2, d0], dim=1),
torch.cat([indices2, indices2], dim=1)))
        d2 = self.dec_conv2(self.upsample2(torch.cat([e1, d1], dim=1),
torch.cat([indices1, indices1], dim=1)))
        d3 = self.dec_conv3(self.upsample3(torch.cat([e0, d2], dim=1),
torch.cat([indices0, indices0], dim=1)))
        return d3

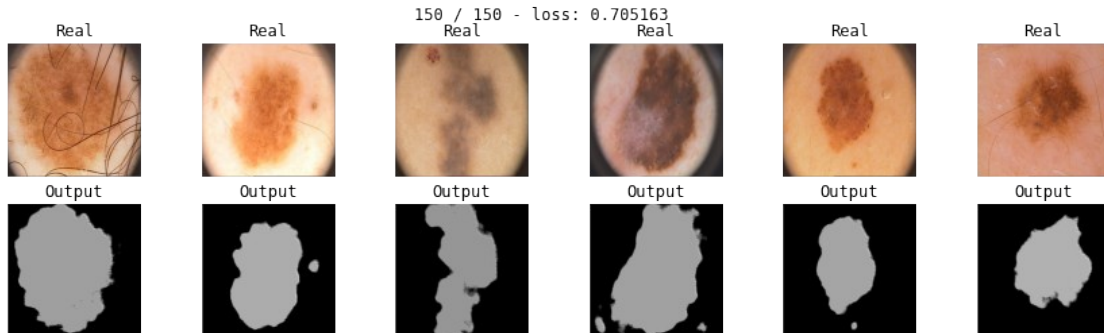
histories = {}
scores = {}
unet_model = UNet().to(device)
max_epochs = 150
opt = optim.AdamW(unet_model.parameters(), lr=1e-4)

```

```

scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
histories = loss_experiment(loss_fn=dice_loss, model=unet_model,
data_tr=data_tr, data_val=data_val, opt=opt,
                           scheduler=scheduler)

```



Epoch 150 train_loss: 0.7052 train_iou: 0.9100 val_loss: 0.7003
val_iou: 0.7360

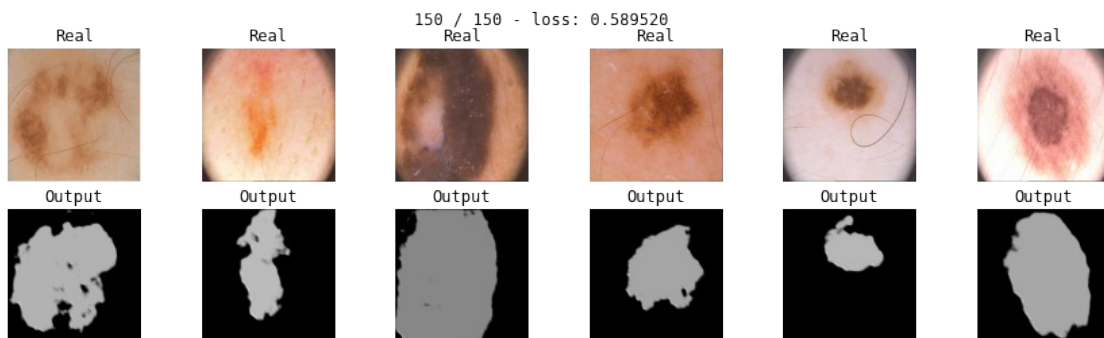
```
score_model(unet_model, iou_pytorch, data_val)
```

0.7700000286102295

```

losses = [bce_loss, focal_loss, tversky_loss, iou_loss]
for loss in losses:
    model_custom = UNet().to(device)
    max_epochs = 150
    opt = optim.AdamW(model_custom.parameters(), lr=1e-4)
    scheduler = optim.lr_scheduler.StepLR(opt, step_size=10,
gamma=0.5)
    loss_experiment(loss_fn=loss, model=model_custom, data_tr=data_tr,
data_val=data_val, opt=opt,
                    scheduler=scheduler)
    print(f"Model with {loss.__name__} :", score_model(model_custom,
iou_pytorch, data_val))

```



Epoch 150 train_loss: 0.5895 train_iou: 0.9680 val_loss: 0.5989

```
val_iou: 0.7840
Model with iou_loss : 0.7840000033378601
```

```
unet_info = {
    'name': "unet",
    'history': histories,
    'scores': scores
}
import json
with open("unet.json", "w") as outfile:
    json.dump(unet_info, outfile)
```

Новая модель путем изменения типа пулинга:

Max-Pooling for the downsampling and **nearest-neighbor Upsampling** for the upsampling.

Down-sampling:

```
conv = nn.Conv2d(3, 64, 3, padding=1)
pool = nn.MaxPool2d(3, 2, padding=1)
```

Up-Sampling

```
upsample = nn.Upsample(32)
conv = nn.Conv2d(64, 64, 3, padding=1)
```

Замените max-pooling на convolutions с stride=2 и upsampling на transpose-convolutions с stride=2.

```
class UNet2(nn.Module):
    def __init__(self):
        super().__init__()

        # encoder (downsampling)
        # Each enc_conv/dec_conv block should look like this:
        # nn.Sequential(
        #     nn.Conv2d(...),
        #     ... (2 or 3 conv layers with relu and batchnorm),
        # )
        self.enc_conv0 = Custom2LayersCNN(in_channels=3,
out_channels=32, kernel_size=3, stride=1, padding=1)
        self.pool0 = torch.nn.Conv2d(32, 32, kernel_size=2, stride=2,
padding=0) # 256 -> 128
        self.enc_conv1 = Custom2LayersCNN(in_channels=32,
out_channels=64, kernel_size=3, stride=1, padding=1)
        self.pool1 = torch.nn.Conv2d(64, 64, kernel_size=2, stride=2,
padding=0) # 128 -> 64
        self.enc_conv2 = Custom3LayersCNN(in_channels=64,
out_channels=128, kernel_size=3, stride=1, padding=1)
        self.pool2 = torch.nn.Conv2d(128, 128, kernel_size=2,
stride=2, padding=0) # 64 -> 32
```



```

        self.enc_conv3 = Custom3LayersCNN(in_channels=128,
out_channels=256, kernel_size=3, stride=1, padding=1)
        self.pool3 = torch.nn.Conv2d(256, 256, kernel_size=2,
stride=2, padding=0) # 32 -> 16

        # bottleneck
        self.bottleneck_conv = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3,
stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3,
stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU()
        )

        # decoder (upsampling)
        self.upsample0 = torch.nn.ConvTranspose2d(256*2, 256*2,
kernel_size=2, stride=2) # 16 -> 32
        self.dec_conv0 = Custom3LayersCNN(in_channels=256*2,
out_channels=128, kernel_size=3, stride=1, padding=1)
        self.upsample1 = torch.nn.ConvTranspose2d(128*2, 128*2,
kernel_size=2, stride=2) # 32 -> 64
        self.dec_conv1 = Custom3LayersCNN(in_channels=128*2,
out_channels=64, kernel_size=3, stride=1, padding=1)
        self.upsample2 = torch.nn.ConvTranspose2d(64*2, 64*2,
kernel_size=2, stride=2) # 64 -> 128
        self.dec_conv2 = Custom3LayersCNN(in_channels=64*2,
out_channels=32, kernel_size=3, stride=1, padding=1)
        self.upsample3 = torch.nn.ConvTranspose2d(32*2, 32*2,
kernel_size=2, stride=2) # 128 -> 256
        self.dec_conv3 = Custom3LayersCNN(in_channels=32*2,
out_channels=1, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        # encoder
        e0 = self.pool0(self.enc_conv0(x))
        e1 = self.pool1(self.enc_conv1(e0))
        e2 = self.pool2(self.enc_conv2(e1))
        e3 = self.pool3(self.enc_conv3(e2))
        # bottleneck
        b = self.bottleneck_conv(e3)

        # decoder
        d0 = self.dec_conv0(self.upsample0(torch.cat([e3, b], dim=1)))
        d1 = self.dec_conv1(self.upsample1(torch.cat([e2, d0],
dim=1)))
        d2 = self.dec_conv2(self.upsample2(torch.cat([e1, d1],
dim=1)))

```

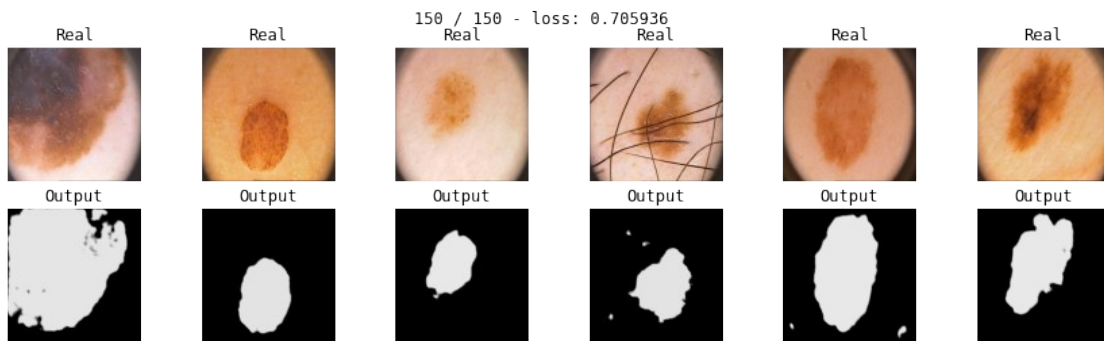


```

        d3 = self.dec_conv3(self.upsample3(torch.cat([e0, d2],
dim=1)))
    return d3

histories = {}
scores = {}
UNET2_model = UNet2().to(device)
max_epochs = 150
opt = optim.AdamW(UNET2_model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
histories = loss_experiment(loss_fn=dice_loss, model=UNET2_model,
data_tr=data_tr, data_val=data_val, opt=opt,
                           scheduler=scheduler)

```



```

Epoch 150 train_loss: 0.7059 train_iou: 0.9200 val_loss: 0.7029
val_iou: 0.7600

```

```

score_model(UNET2_model, iou_pytorch, data_val)

```

```

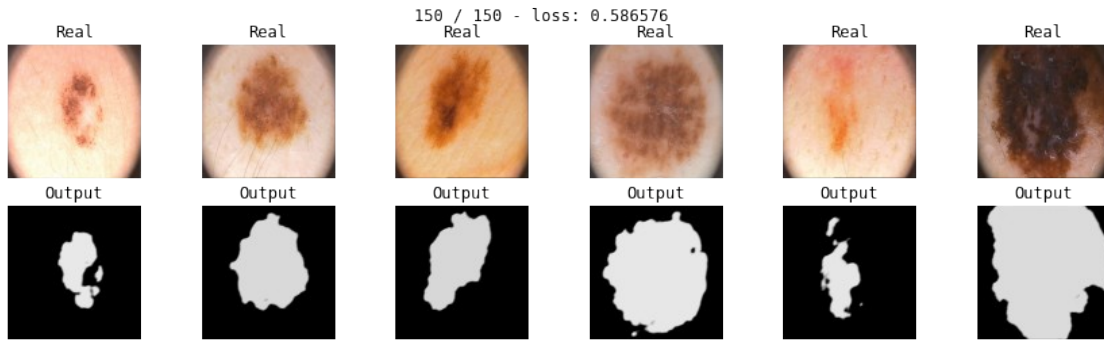
0.7200000286102295

```

```

losses = [bce_loss, focal_loss, tversky_loss, iou_loss]
for loss in losses:
    model_custom = UNet2().to(device)
    max_epochs = 150
    opt = optim.AdamW(model_custom.parameters(), lr=1e-4)
    scheduler = optim.lr_scheduler.StepLR(opt, step_size=10,
gamma=0.5)
    loss_experiment(loss_fn=loss, model=model_custom, data_tr=data_tr,
data_val=data_val, opt=opt,
                    scheduler=scheduler)
    print(f"Model with {loss.__name__} :", score_model(model_custom,
iou_pytorch, data_val))

```



Epoch 150 train_loss: 0.5866 train_iou: 0.9770 val_loss: 0.5980
 val_iou: 0.7580
 Model with iou_loss : 0.7580000519752502

```

UNET2_INFO = {
    'name': "UNET",
    'history': histories,
    'scores': scores
}
import json
with open("UNET2.json", "w") as outfile:
    json.dump(UNET2_INFO, outfile)

```

```

from google.colab import files
files.download('segnet.json')
files.download('UNET.json')
files.download('UNET2.json')
print("Completed")

```

<IPython.core.display.Javascript object>
 <IPython.core.display.Javascript object>
 <IPython.core.display.Javascript object>
 <IPython.core.display.Javascript object>
 <IPython.core.display.Javascript object>
 <IPython.core.display.Javascript object>

Completed

```

with open('segnet.json') as json_file:
    segnet = json.load(json_file)
with open('UNET_1.json') as json_file:
    UNET = json.load(json_file)
with open('UNET2_1.json') as json_file:
    UNET2 = json.load(json_file)

```

Сделайте вывод, какая из моделей лучше

Отчет (6 баллов):

Ниже предлагается написать отчет о проделанной работе и построить графики для лоссов, метрик на валидации и тесте. Если вы пропустили какую-то часть в задании выше, то вы все равно можете получить основную часть баллов в отчете, если правильно зададите проверяемые вами гипотезы.

Аккуратно сравните модели между собой и соберите наилучшую архитектуру. Проверьте каждую модель с различными лоссами. Мы не ограничиваем вас в формате отчета, но проверяющий должен отчетливо понять для чего построен каждый график, какие выводы вы из него сделали и какой общий вывод можно сделать на основании данных моделей. Если вы захотите добавить что-то еще, чтобы увеличить шансы получения максимального балла, то добавляйте отдельное сравнение.

Дополнительные комментарии:

Пусть у вас есть N обученных моделей.

- Является ли отчетом N графиков с 1 линией? Да, но очень низкокачественным, потому что проверяющий не сможет сам сравнить их.
- Является ли отчетом 1 график с N линиями? Да, но скорее всего таким образом вы отразили лишь один эффект. Этого мало, чтобы сделать достаточно суждений по поводу вашей работы.
- Я проверял метрики на трейне, и привел в результате таблицу с N числами, что не так? Ключевой момент тут, что вы измеряли на трейне ваши метрики, уверены ли вы, что зависимости останутся такими же на отложенной выборке?
- Я сделал отчет содержащий график лоссов и метрик, и у меня нет ошибок в основной части, но за отчет не стоит максимум, почему? Естественно максимум баллов за отчет можно получить не за 2 графика (даже при условии их полной правильности). Проверяющий хочет видеть больше сравнений моделей, чем метрики и лоссы (особенно, если они на трейне).

Советы: попробуйте правильно поставить вопрос на который вы себе отвечаете и продемонстрировать таблицу/график, помогающий проверяющему увидеть ответ на этот вопрос. Пример: Ваня хочет узнать, с каким из 4-х лоссов модель (например, U-Net) имеет наилучшее качество. Что нужно сделать Ване? Обучить 4 одинаковых модели с разными лосс

функциями. И измерить итоговое качество. Продемонстрировать результаты своих измерений и итоговый вывод. (warning: конечно же, это не идеально ответит на наш вопрос, так как мы не учитываем в экспериментах возможные различные типы ошибок, но для первого приближения этого вполне достаточно).

Примерное время на подготовку отчета 1 час, он содержит сравнение метрик, график лоссов, выбор лучших моделей из нескольких кластеров и выбор просто лучшей модели, небольшой вывод по всему дз, возможно сравнение результирующих сегментаций, времени или числа параметров модели, проявляйте креативность.

```
def plot_val_metrics(histories: list):
    plt.style.use('seaborn-dark-palette')
    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(20,6))
    for loss_fn_name, history in histories.items():
        loss, acc, val_loss, val_acc = zip(*history)
        ax.plot(val_acc, label=f'val_{loss_fn_name}_iou')
        ax.set_title('Intersection over Union (IoU)')

        ax.legend(loc='best')

        ax.set_xlabel("epochs")
        ax.set_ylabel("iou")
    plt.show()

import copy
def print_mask(data: Iterable, model: nn.Module):
    new_data = copy.deepcopy(data)
    X_test, Y_test = next(iter(new_data))
    model.eval()
    Y_hat = model(X_test.to(device)).detach().cpu().numpy()
    for k in range(6):
        plt.subplot(2, 6, k+1)
        plt.imshow(np.rollaxis(X_test[k].numpy(), 0, 3), cmap='gray')
        plt.title('Real')
        plt.axis('off')
        plt.subplot(2, 6, k+7)
        plt.imshow(Y_hat[k, 0], cmap='gray')
        plt.title('Output')
        plt.axis('off')
    plt.show()

def create_barplot(data: dict, title: str='', figsize: tuple=(12,6)):
    fig, ax = plt.subplots(figsize=figsize)
    color_rectangle = ['gray', 'deepskyblue', 'indigo',
                       'crimson', 'hotpink']
    ax.bar(data.keys(), data.values(),
           alpha=0.5,
           color=color_rectangle,
           edgcolor=color_rectangle, # цвет края прямоугольника
```

```

        linewidth=4,          # ширина крайней линии
        linestyle='--',
        width=0.8)
labels = [round(v,3) for v in list(data.values())]
for i, v in enumerate(labels):
    ax.text(i-0.1,
            v/2,
            labels[i],
            fontsize=10,
            color=color_rectangle[i])
ax.set_facecolor('seashell')

plt.show()

```

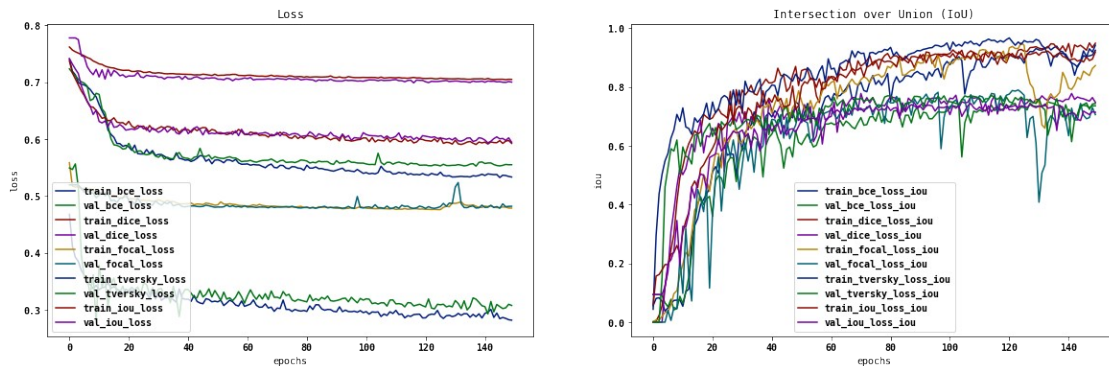
Losses and Metrics

```

from IPython.display import display, Markdown
display(Markdown('<h3>SegNet</h3>'))
plot_histories(segnet_info['history'])

```

<IPython.core.display.Markdown object>

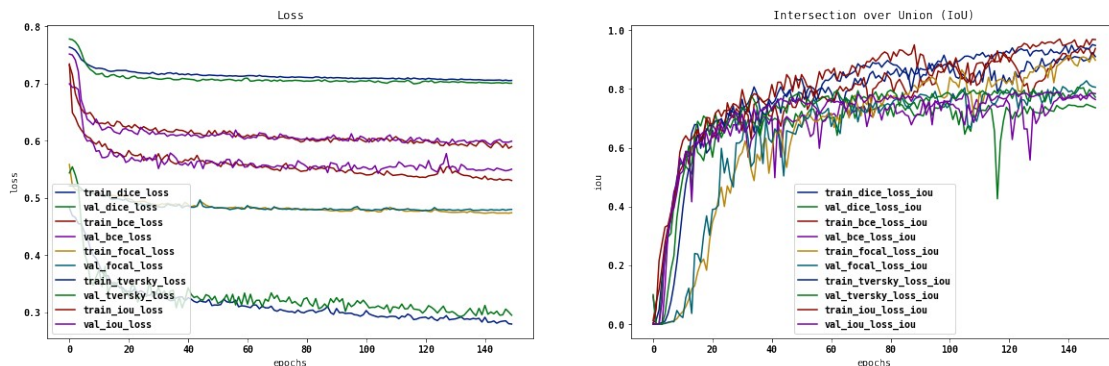


```

display(Markdown('<h3>UNet</h3>'))
plot_histories(unet_info['history'])

```

<IPython.core.display.Markdown object>

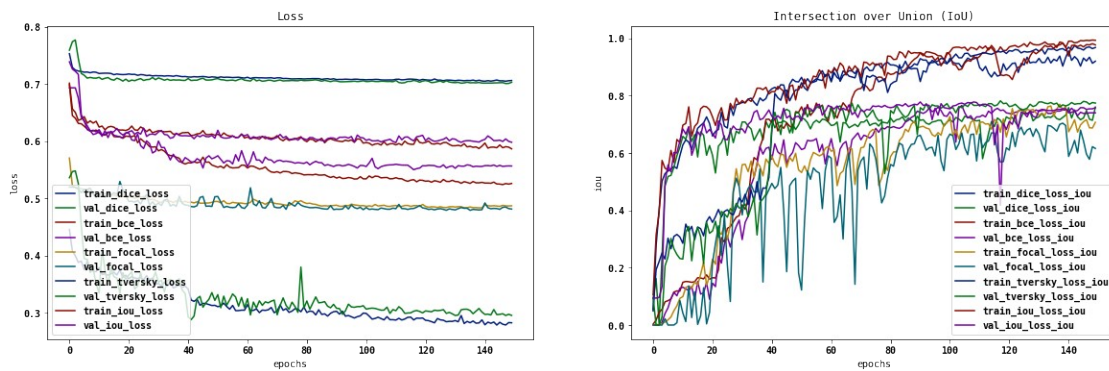


```

display(Markdown('<h3>UNet with changed pool type</h3>'))
plot_histories(unet2_info['history'])

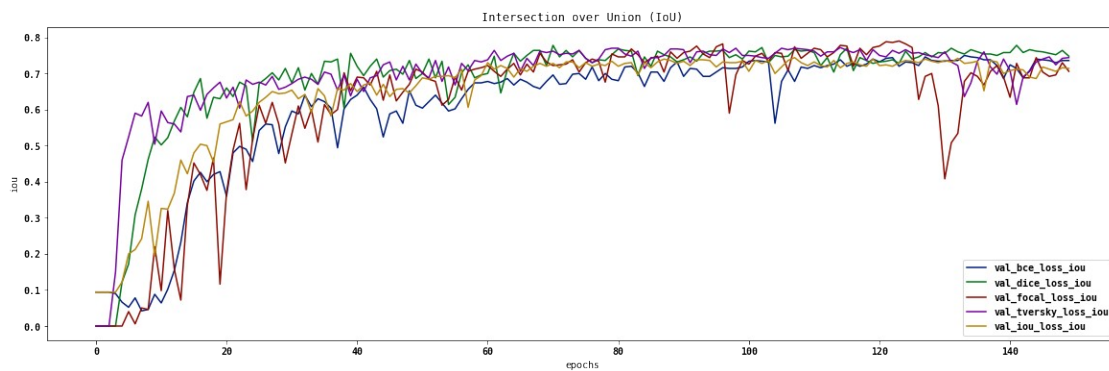
```

<IPython.core.display.Markdown object>

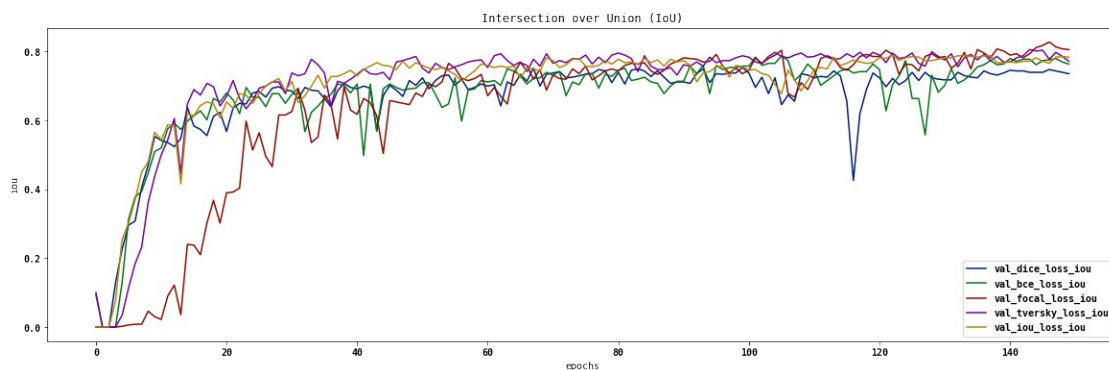


```
display(Markdown('<h3>SegNet validation metrics</h3>'))
plot_val_metrics(segnet_info['history'])
display(Markdown('<h3>UNet validation metrics</h3>'))
plot_val_metrics(unet_info['history'])
display(Markdown('<h3>UNet with changed pool types validation
metrics</h3>'))
plot_val_metrics(unet2_info['history'])
```

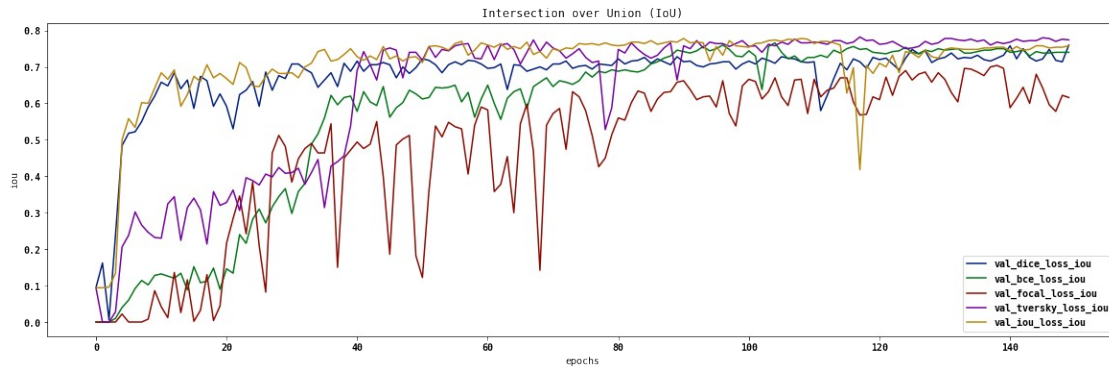
<IPython.core.display.Markdown object>



<IPython.core.display.Markdown object>



<IPython.core.display.Markdown object>

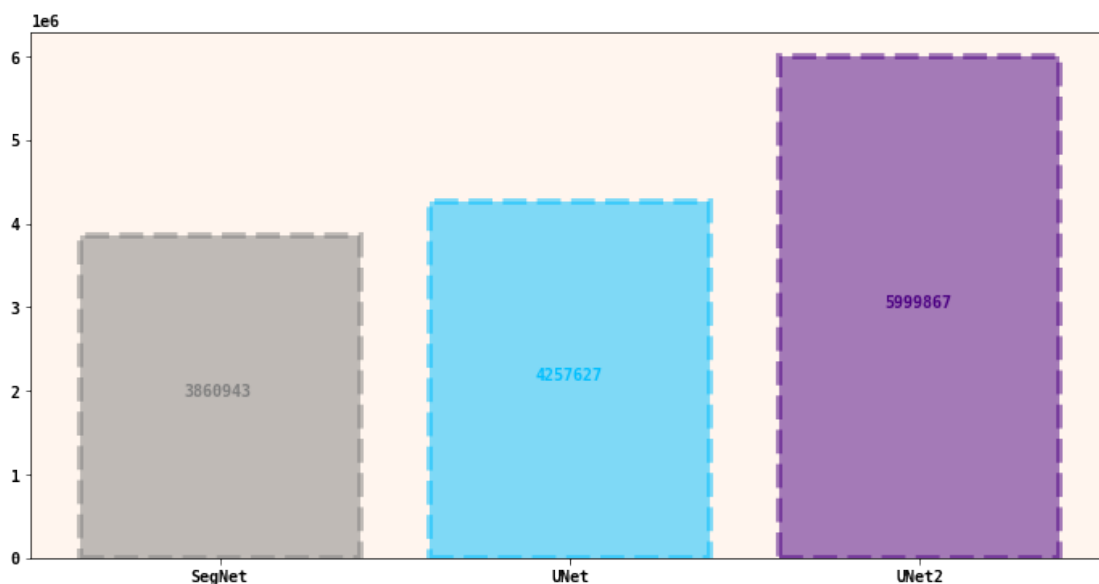


В целом, все обучение со всеми функциями ошибки находится в одном диапазоне метрик. Видно, что к 100й эпохе начинается переобучение, что связано, полагаю, в том числе и с маленькой выборкой.

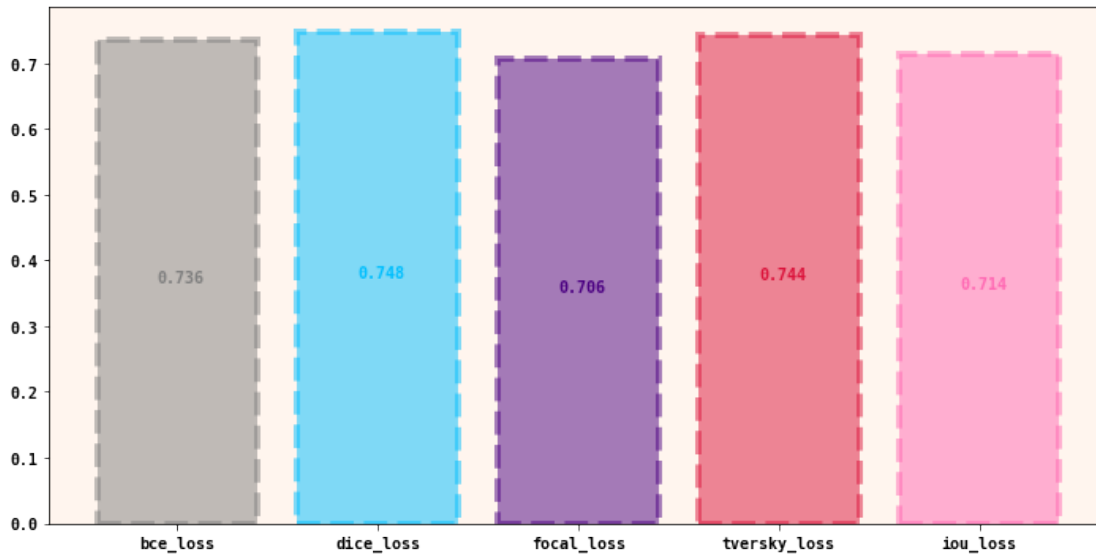
```
trainable_parameters_count = {}
segnet_model = SegNet().to(device)
trainable_parameters_count[SegNet.__name__] = sum(p.numel() for p in
segnet_model.parameters() if p.requires_grad)
unet_model = UNet().to(device)
trainable_parameters_count[UNet.__name__] = sum(p.numel() for p in
unet_model.parameters() if p.requires_grad)
unet2_model = UNet2().to(device)
trainable_parameters_count[UNet2.__name__] = sum(p.numel() for p in
unet2_model.parameters() if p.requires_grad)
```

```
display(Markdown('<h3>Comparing models by the number of trainable
parameters</h3>'))
create_barplot(trainable_parameters_count)
```

<IPython.core.display.Markdown object>

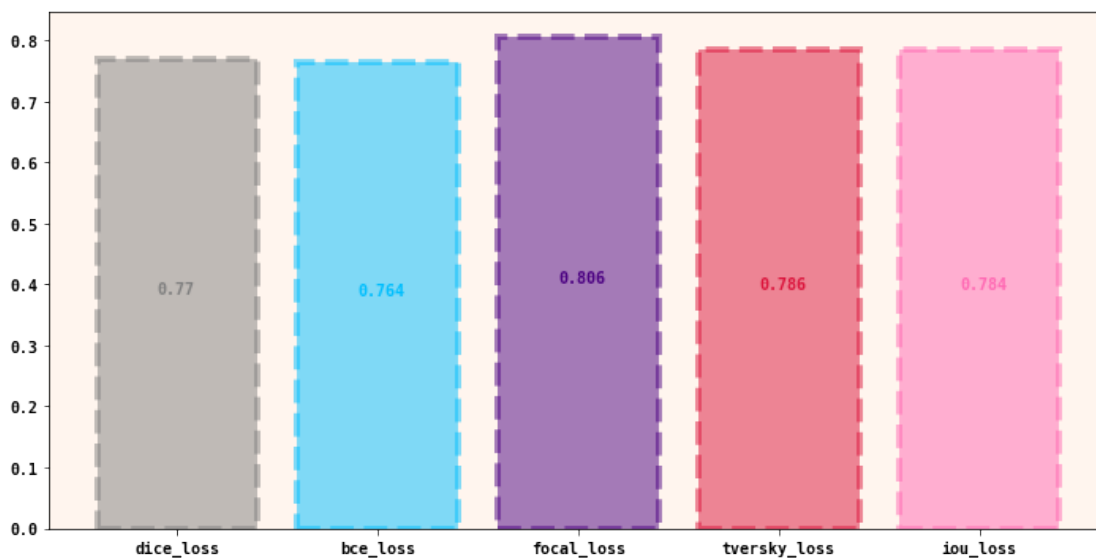


```
display(Markdown('<h3>SegNet different losses validation metrics</h3>'))
create_barplot(segnet_info['scores'])
<IPython.core.display.Markdown object>
```



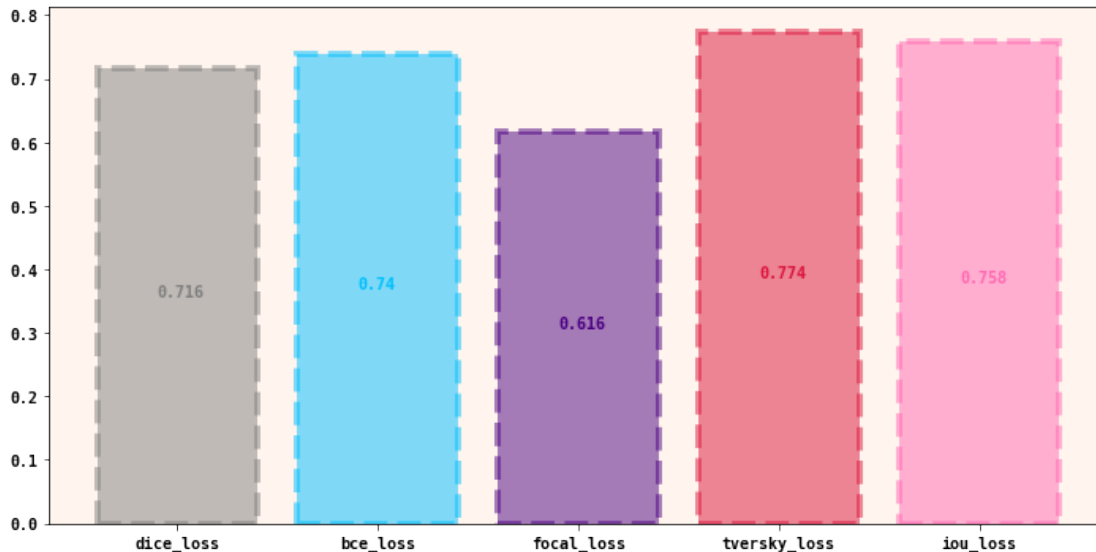
Возьмем SegNet с dice_loss

```
display(Markdown('<h3>UNet different losses validation metrics</h3>'))
create_barplot(unet_info['scores'])
<IPython.core.display.Markdown object>
```



UNet с focal_loss

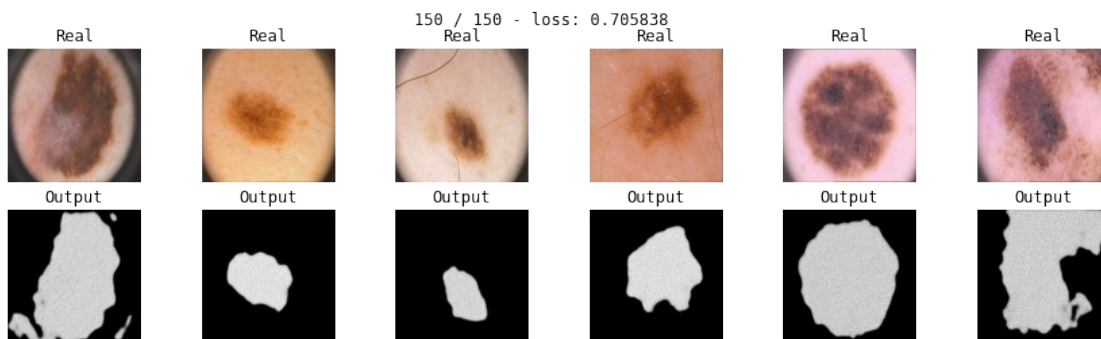

```
display(Markdown('<h3>SegNet different losses validation  
metrics</h3>'))
create_barplot(unet2_info['scores'])
<IPython.core.display.Markdown object>
```



И UNet с измененным типом пуллинга с tversky_loss

Masks by the best models

```
max_epochs = 150
opt = optim.AdamW(segnet_model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
loss_experiment(loss_fn=dice_loss, model=segnet_model,
data_tr=data_tr, data_val=data_val, opt=opt,
scheduler=scheduler);
```



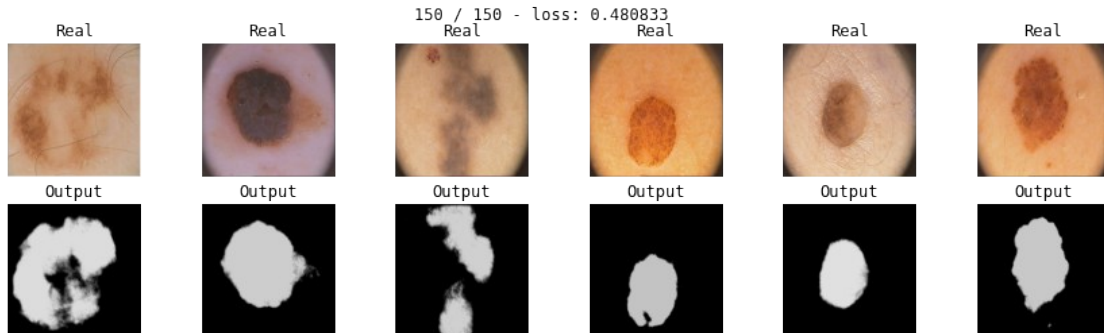
Epoch 150 train_loss: 0.7058 train_iou: 0.9030 val_loss: 0.7038
val_iou: 0.7300

```
max_epochs = 150
opt = optim.AdamW(unet_model.parameters(), lr=1e-4)
```

```

scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
loss_experiment(loss_fn=focal_loss, model=unet_model, data_tr=data_tr,
data_val=data_val, opt=opt,
                scheduler=scheduler);

```

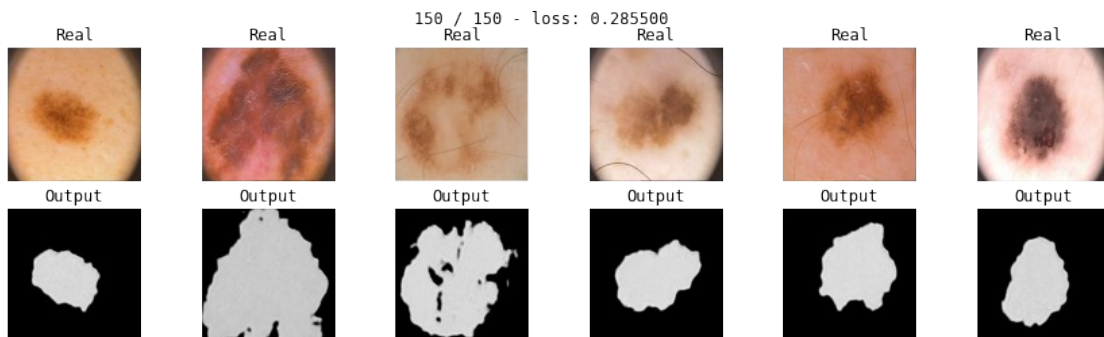


Epoch 150 train_loss: 0.4808 train_iou: 0.8200 val_loss: 0.4789
val_iou: 0.7520

```

unet2_model = UNet2().to(device)
max_epochs = 150
opt = optim.AdamW(unet2_model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=10, gamma=0.5)
histories = loss_experiment(loss_fn=tversky_loss, model=unet2_model,
data_tr=data_tr, data_val=data_val, opt=opt,
                scheduler=scheduler)

```



Epoch 150 train_loss: 0.2855 train_iou: 0.9800 val_loss: 0.2936
val_iou: 0.7640

```

def print_mask(data: Iterable, models: list):
    X_test, Y_test = next(iter(data))
    for model in models:
        model.eval()
        Y_hat = model(X_test.to(device)).detach().cpu().numpy()
        display(Markdown(f'<h3>{type(model).__name__} mask
predictions</h3>'))
        for k in range(6):

```

```

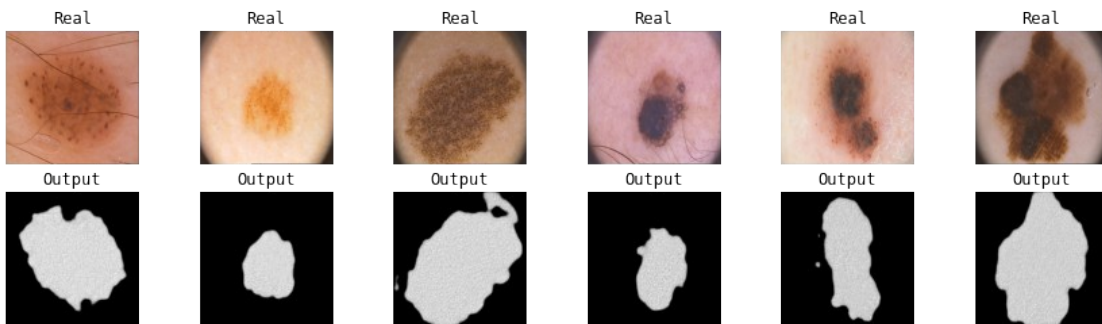
plt.subplot(2, 6, k+1)
plt.imshow(np.rollaxis(X_test[k].numpy(), 0, 3),
cmap='gray')
plt.title('Real')
plt.axis('off')
plt.subplot(2, 6, k+7)
plt.imshow(Y_hat[k, 0], cmap='gray')
plt.title('Output')
plt.axis('off')
plt.show()

```

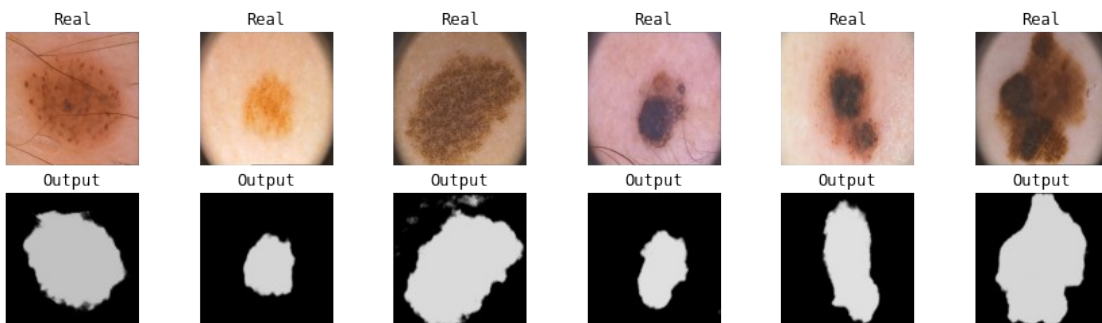
```
models = [segnet_model, unet_model, unet2_model]
```

```
print_mask(data_ts, models)
```

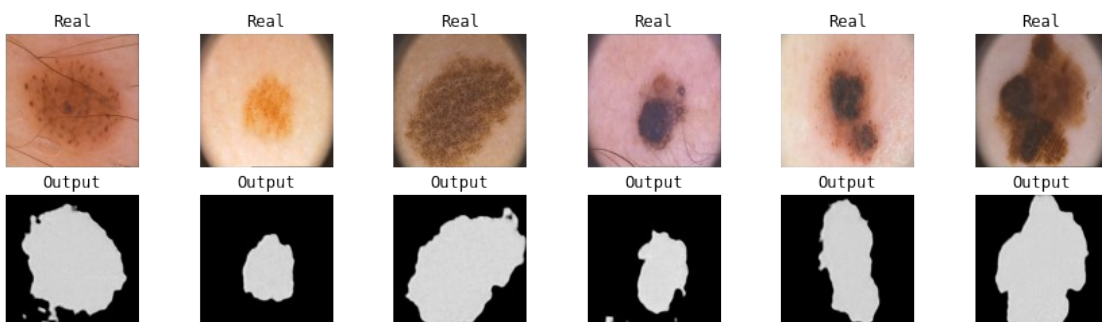
<IPython.core.display.Markdown object>



<IPython.core.display.Markdown object>



<IPython.core.display.Markdown object>



```
scores = {  
    'SegNet': score_model(segnet_model, iou_pytorch, data_ts),  
    'UNet': score_model(unet_model, iou_pytorch, data_ts),  
    'UNet with changed pool type': score_model(unet2_model,  
iou_pytorch, data_ts)  
}  
display(Markdown(f'<h1>Metrics on the test dataset</h1>'))  
create_barplot(scores)
```

<IPython.core.display.Markdown object>

